The Morphware Stable Interface: A Software Framework for Polymorphous Computing Architectures

Daniel P. Campbell
Mark A. Richards
Georgia Institute of Technology
Atlanta, GA
770-528-7541 / dan.campbell@gtri.gatech.edu

Dennis M. Cottel
Randall R. Judd
SPAWAR Systems Center
San Diego, CA

Kenneth M. Mackenzie
Reservoir Labs, Inc.
New York, NY

Abstract

Polymorphous Computing Architectures (PCAs) are computing devices that are capable of significant, rapid reconfiguration directed by software. Composed of several groups of computing elements, PCA devices can be configured to achieve high performance on a wide variety of problem types and processing demands. We describe an emerging concept called the Morphware Stable Interface (MSI), a portable, scalable software development framework to harness the power and complexity of PCAs, while allowing productive software development, and rapid adoption of PCA devices.

1. INTRODUCTION

The Polymorphous Computing Architectures (PCA) program is a Defense Advanced Research Projects Agency (DARPA) effort to develop new embedded computing platforms with very strong, rapid in-mission reconfigurability. Target applications range from military platforms that must adapt to rapidly changing mission parameters, to embedded network controllers, whose optimal configuration of hardware resources will change in response to the traffic and environmental conditions it faces.

The PCA program "core projects" working to develop microprocessors that implement polymorphous capabilities include Smart Memories [1], Raw [2], M3T [3], TRIPS [4], and MONARCH [5]. The chips under development in these projects have several characteristics in common. These are typically tiled structures composed from replicated, fully capable computing cores, reconfigurable memory and cache elements, and a rich set of reconfigurable data paths, network interfaces, and I/O paths. Each can operate in streaming or threaded modes. Each has mechanisms for aggregating individual processor tiles into larger compound processor units. They differ in their approach for aggregating processors and in their emphasis on processor, memory, or communication design. Figure 1 illustrates a generic PCA microarchitecture.

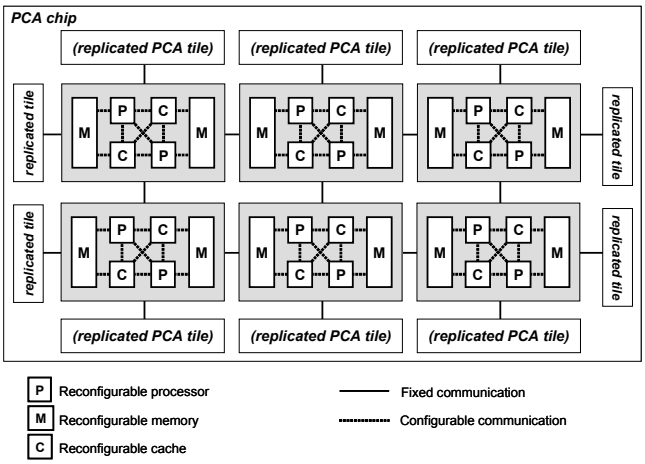


Figure 1. Generic PCA chip micro-architecture.

The increased capability of PCA systems comes at the expense of increased software complexity. Applications written with knowledge about the platform embedded into the structure of the application can make use of the reconfigurability of such resources, but suffer from a lack of scalability and portability. Applications written with no such knowledge are completely dependent on build and run-time systems to utilize the capabilities of reconfigurable systems.

An important goal of the PCA program is therefore to create an application development framework, called the Morphware Stable Interface (MSI), that will exploit the capabilities of PCA hardware while retaining as much portability and performance as possible. The MSI should:

- Support dynamic hardware reorganization and optimization
- Obtain nearly optimal performance
- Abstract configurable computing elements
- Abstract hardware reconfiguration
- Mitigate development and runtime complexity
- Leverage existing technologies.

This paper presents the early results of the MSI design effort.

2. THE MORPHWARE FORUM

To facilitate the development of the cross-project consensus and design effort needed to realize the MSI, the PCA program has formed the Morphware Forum, an informal consortium of the PCA contractors and other selected participants. Organized and led by Georgia Tech under DARPA sponsorship, the Forum meets quarterly, with other interim activities as required, to develop and debate proposals for the MSI. The Forum will ultimately develop a set of publicly available standards documents that define the architecture and details of the MSI.

Georgia Tech maintains a Morphware Forum web site at http://www.morphware.org that provides a vehicle for meeting planning, collaborative exchange, and public information about the MSI effort. At this writing, the public portion of the site is limited to introductory papers and briefings on the MSI effort and the PCA systems, and links to program participants' web sites. MSI standards documents will be available at this site as the Forum approves them.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
20 AUG 2004	**N/A**	**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
The Morphware Stable Interface: A Software Framework for Polymorphous Computing Architectures	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Georgia Institute of Technology Atlanta, GA; SPAWAR Systems Center, San Diego, CA; Reservoir Labs, Inc. New York, Inc**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
See also ADM001694, HPEC-6-Vol 1 ESC-TR-2003-081; High Performance Embedded Computing (HPEC) Workshop (7th)., The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**UU**	**27**	

3. MORPHWARE STABLE INTERFACE

The MSI is a framework consisting of several major elements. Some of the most significant aspects of the MSI are described.

3.1 The Streaming Virtual Machine

Several of the projects have developed specialized languages to exploit the differences between PCAs and traditional CPUs. The highly parallelized, low unit-to-unit latency of PCA devices exposes a need for compilers to more fully understand data dependencies and control flow within an application than is possible with languages such as C. In order to expose these elements more clearly to compilers, the projects have developed variants of C and C++ that introduce and enforce data-stream, and operational kernel constructs. These language express programs as directed data flow graphs connecting various computational kernels. The restrictions on data and control flow allow compilers to map algorithms very effectively to PCA-style devices, while simplifying source code for a wide range of applications. These languages express a fundamentally different underlying virtual machine than languages such as C, and form an important aspect of the MSI.

3.2 MSI Portability Layers

From its inception, the PCA program has advocated dual portability layers for the MSI. The MSI provides portability via the upper "Stable API" (SAPI) and the lower "Stable Architecture Abstraction Layer" (SAAL). Figure 2 illustrates the concept. It reduces development effort for application build systems by providing a common middle layer; allows addition of new top level application approaches without breaking existing build systems; provides a stable, platform-independent target for top level build tools; allows dynamic compilation, and increases specialization opportunities in the builder/middleware marketplace.

The SAPI consists of multiple platform-independent, high-level languages and metadata representations. These languages will be extensions to existing languages such as C or C++ that express each of the virtual machine abstractions present in the SAAL. The SAAL will be a portable C-based representation of a virtual machine with both streaming and threaded modes. It will be both platform- and SAPI language-independent. Finally, it will be lower level than SAPI languages, and able to support dynamic compilation. The Morphware Forum is currently actively developing detailed proposals for the SAAL VM.

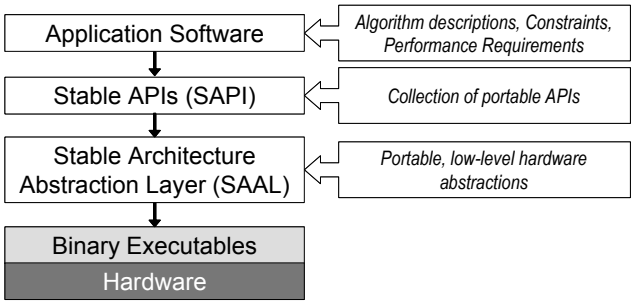


Figure 2. SAPI and SAAL MSI portability layers.

3.3 Component-Based Framework

The ability of applications to intelligently reconfigure host platforms is a critical element of PCA software. In order to facilitate morphing, a component-based software framework has been proposed. Components provide natural and intuitive boundaries for run-time reconfiguration of hardware. Software components will be built for varying hardware configurations, and the appropriate version may be loaded and executed on the fly as portions of the host platform are reconfigured by the PCA system or application software.

3.4 Metadata

PCA systems must to be aware of and reactive to application, resource, and constraint goals and requirements. Examples include SWEPT (size, weight, energy, performance, and time) constraints, quality of service requirements (*e.g.* latency, throughput), morph policy, security requirements, and so forth. Metadata provides a means for the MSI to represent the information needed to implement this capability.

Metadata are non-functional descriptions of requirements, constraints, desired resource management policy, hardware configuration options or any other information that expresses information about system operation independent of the application functional description. Each of the uses of metadata constitutes a unique context that must be standardized and described. A standard method for describing metadata contexts has been proposed and is currently under consideration by the Morphware Forum. Specifications for several metadata contexts have been proposed, and standard appropriate storage, retrieval, and query mechanisms for the metadata are currently being designed.

4. REFERENCES

[1] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. Dally, and M. Horowitz, "Smart Memories: A Modular Reconfigurable Architecture", *Proceedings* International Symposium on Computer Architecture, June 2000.

[2] M. B. Taylor, J. Kim, *et al.*, "The Raw Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs", IEEE *Micro*, April/March 2002.

[3] C. Cascaval *et al*, "Evaluation of a Multithreaded Architecture for Cellular Computing", *Proceedings* Eighth International Symposium on High-Performance Computer Architecture (HPCA), February 2002.

[4] R. Nagarajan *et al*, "A Design Space Evaluation of Grid Processor Architectures", *Proceedings* 34th Annual International Symposium on Microarchitecture, pp.40-51, December, 2001.

[5] J. Granacki and M. Vahey, "MONARCH: A Morphable Networked micro-ARCHitecture", High Performance Embedded Computing Workshop, October 2002.

The Morphware Stable Interface: A Software Framework for Polymorphous Computing Architectures

D. Campbell[1], D. Cottel[2], R. Judd[2], K. MacKenzie[3], M. Richards[4]

[1]Georgia Tech Research Institute, Smyrna, GA
[2]U.S. Navy SPAWAR Systems Center, San Diego, CA
[3]Reservoir Labs, Inc. New York, NY
[4]Georgia Institute of Technology, Atlanta, GA

Georgia Tech Research Institute

SPAWAR

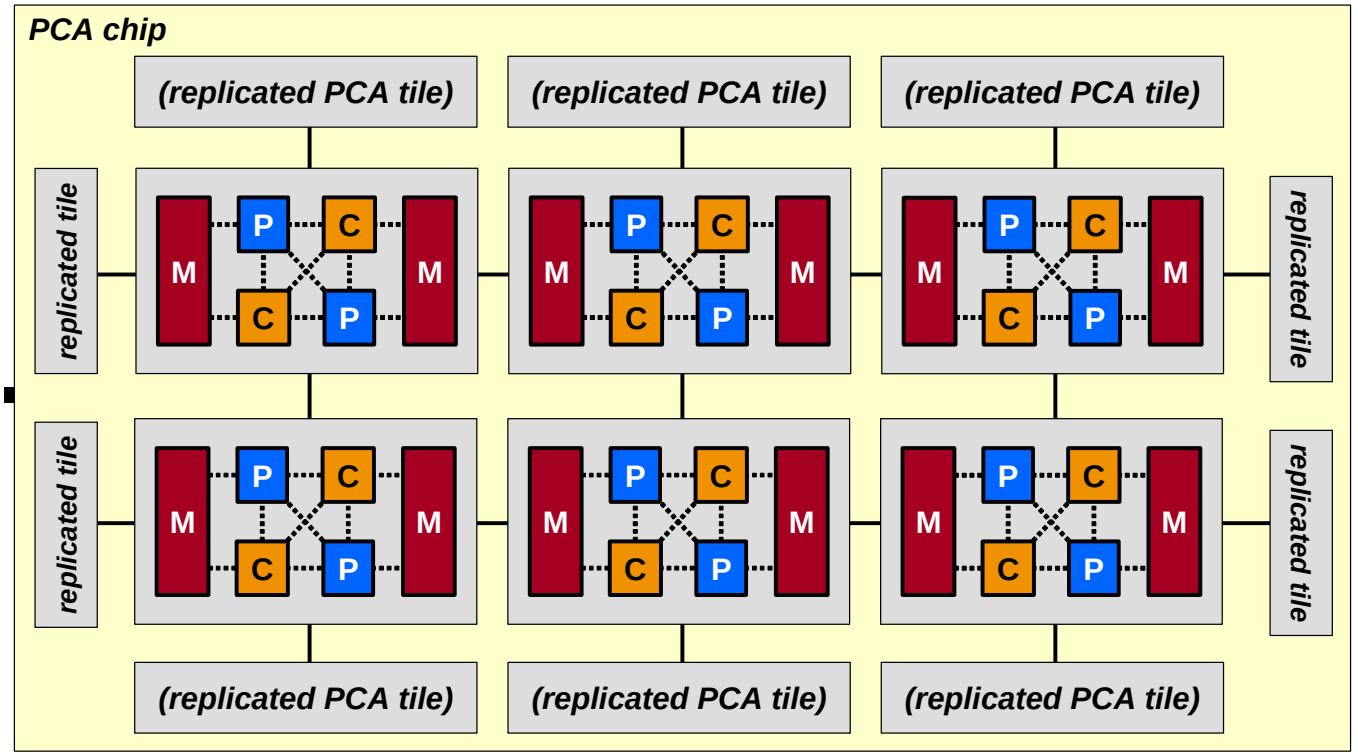
reservoir labs®

Acknowledgements

Polymorphous Computing Architectures

- **DARPA effort for high performance embedded platforms with strong, rapid, reactive in-mission configurability**
  - **Support dynamic and multi-mission requirements**
  - **Support collaborative, information-centric missions**
- **PCA will develop processing architectures that "morph"**
  - **Hardware and software resources reconfigure to balance resource requirements and availability**
    - **at multiple levels: micro-architecture, network, system**
    - **at multiple time scales: in-mission, between-mission**

Generic PCA Microarchitecture



Generic PCA Microarchitecture

- **Tiled structure**
  - Fully capable computing cores
  - Configurable memory and cache
  - Configurable data paths, network interfaces, and I/O
  - Streaming and Threaded modes
  - Methods to aggregate tiles into larger processing units
- **Core projects differ in**
  - aggregation mechanisms
  - relative emphasis on processor, memory, or comm design
- **Performance on the order of (per chip)**
  - 4 – 64 GFLOPS / 4 – 16 GOPS
  - 25 – 32 GB/s off-chip I/O

Software and PCA

- **Increased hardware flexibility and complexity brings increased software complexity**
  - **If we build target platform reconfigurability and performance info into the application, we lose scalability and portability**
  - **If we don't, the build and run-time systems will be entirely responsible for leveraging the platform capability, and we still lose fine-grain morphability**
  - **Applications must be reactive to feedback from the hardware**
    - **resource collisions, SWEPT, faults**
- **Solution: the Morphware Stable Interface (MSI)**

The Morphware Stable Interface (MSI)

- **Application Development Framework for PCAs**
- **Comprised of a software architecture and a suite of open standard APIs**
- **Goals**
  - **Dynamically optimize PCA resources for application functionality, service requirements, and constraints**
  - **Obtain nearly optimal performance from PCA hardware**
  - **Be highly reactive to PCA hardware and user inputs**
  - **Manage PCA software complexity**
  - **Leverage existing and already-developing technologies**
- **Cross-project effort, developed in parallel with the hardware**

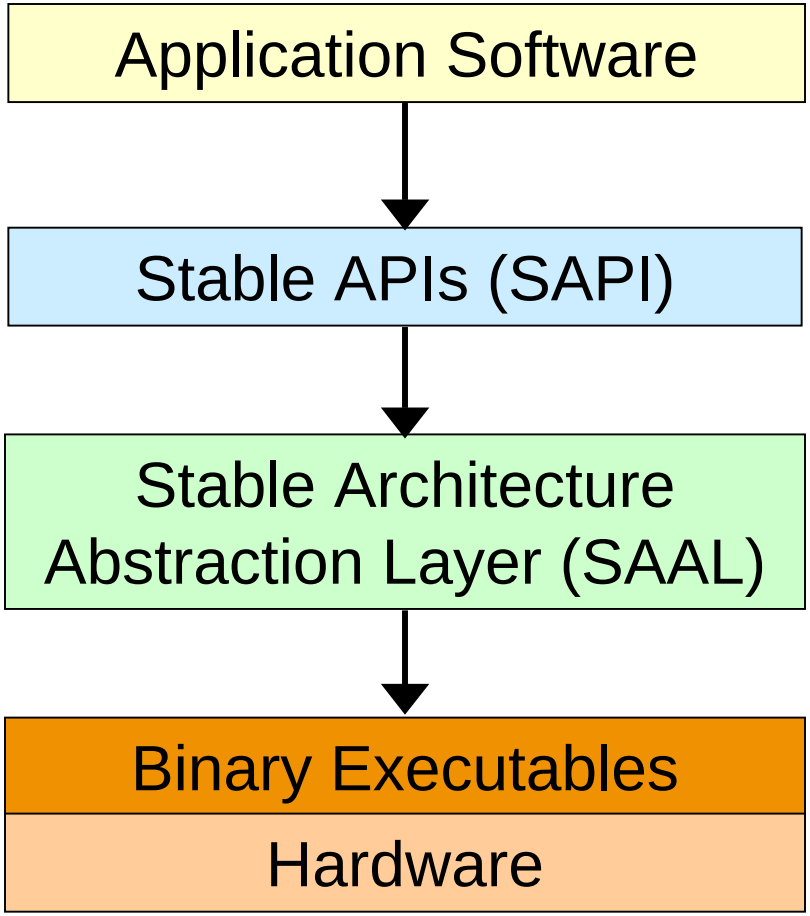
The Morphware Forum

- Informal consortium of the PCA contractors and other selected participants
- Organized and led by the Georgia Tech/SPAWAR team
- Meets quarterly
  - interim meetings and activities as required
- Propose, debate, develop, test, validate, document, and demonstrate standards that define the MSI

The Morphware Forum

- Applied Photonics
- Georgia Institute of Technology
- George Mason University
- IBM
- Lockheed Martin Company
- Massachusetts Institute of Technology
- MIT/Lincoln Laboratory
- Mercury Computing Systems
- Mississippi State University
- MPI Software Technology, Inc.
- Northrup Grumman
- Reservoir Labs, Inc.
- Raytheon
- SPAWAR
- South West Research
- Stanford University
- University of Texas - Austin
- University of Illinois
- University of Pennsylvania
- University of Southern California
- Vanderbilt University

Dual Portability Layers



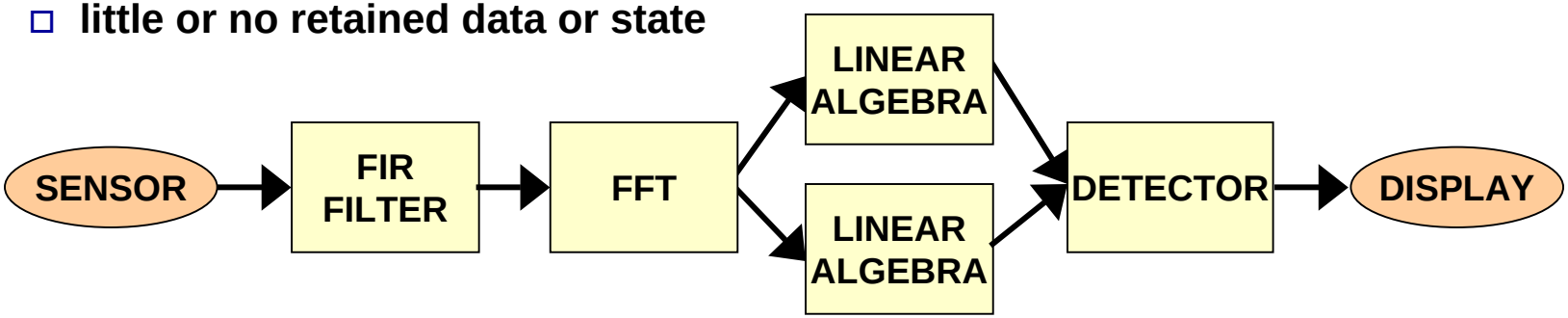
- **Stable API (SAPI) and Stable Architecture Abstraction Layer (SAAL) provide dual portability layers**
- **Application SW describes functionality, constraints, and performance requirements**
- **SAPI is PCA-aware collection of standardized language and service APIs**
- **SAAL is PCA-aware abstracted low-level machine representations**

Why SAAL?

- Traditional languages based on a machine model increasingly incorrect
  - Single program counter
  - One operation at a time
  - Data universally local
- All modern high performance computing systems battling this issue
- In order to exploit new hardware, core teams developed new languages not based on old model
- New languages based on similar models
- Formalize the models to make it explicit

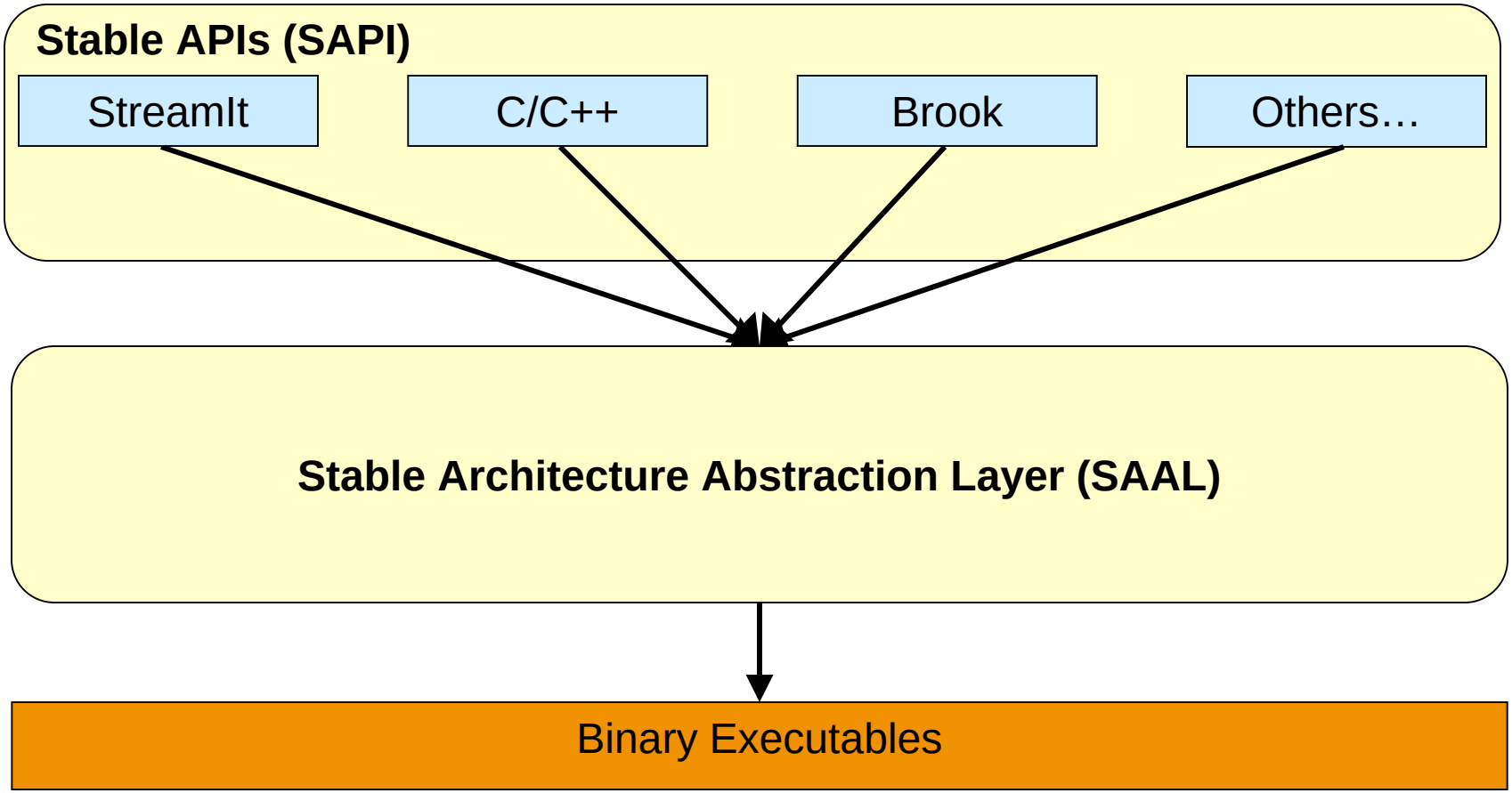
Stream Languages

- Compute-intensive portions of many applications have characteristics of stream operations
  - fixed data flow graph
  - large, possibly infinite, data stream
  - functional kernels not data-dependent
  - functional kernels independent of one another
  - little or no retained data or state



- Representations that enforce these characteristics ideally suit PCA architectures, aid compiler in
  - Optimization
  - Scheduling
  - Resource allocation
  - Data Locality

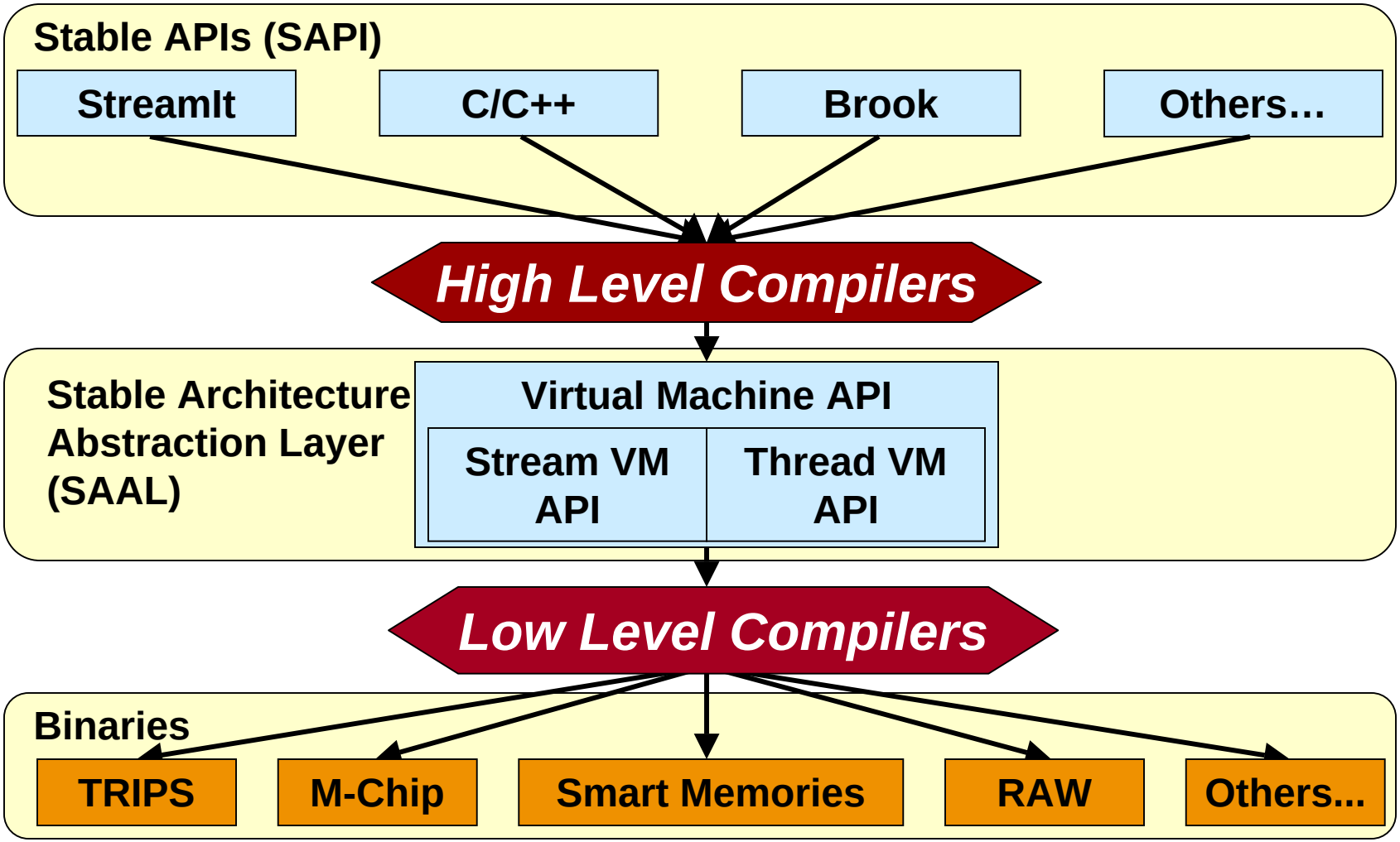
Morphware Languages



SAAL Instantiation

- Traditional languages have an implicit SAAL layer
- MSI has an explicit SAAL layer, a portable API that exposes the virtual resources typical of PCA systems
  - Sacrifices some tool chain flexibility for simpler, more defined, more analyzable build chain
  - Factors deployment of new languages and hardware
  - Allows explicit consideration of model of computer
  - Formalizes and augments existing model
- Creates a two-stage compile process
- Example constructs: kernel, stream, processor, etc

Morphware Compilation

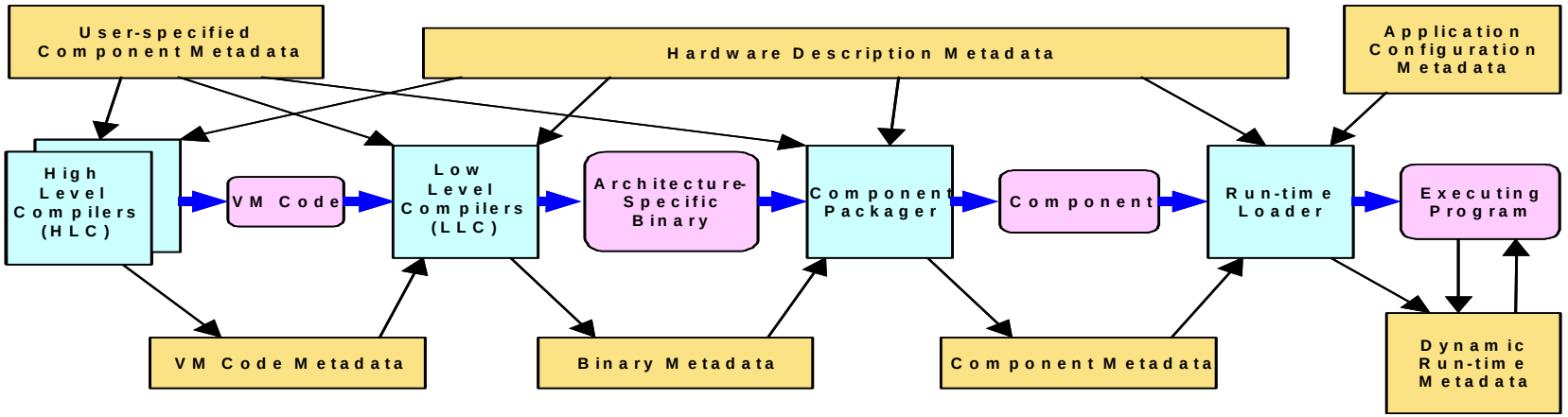


Metadata in Morphware

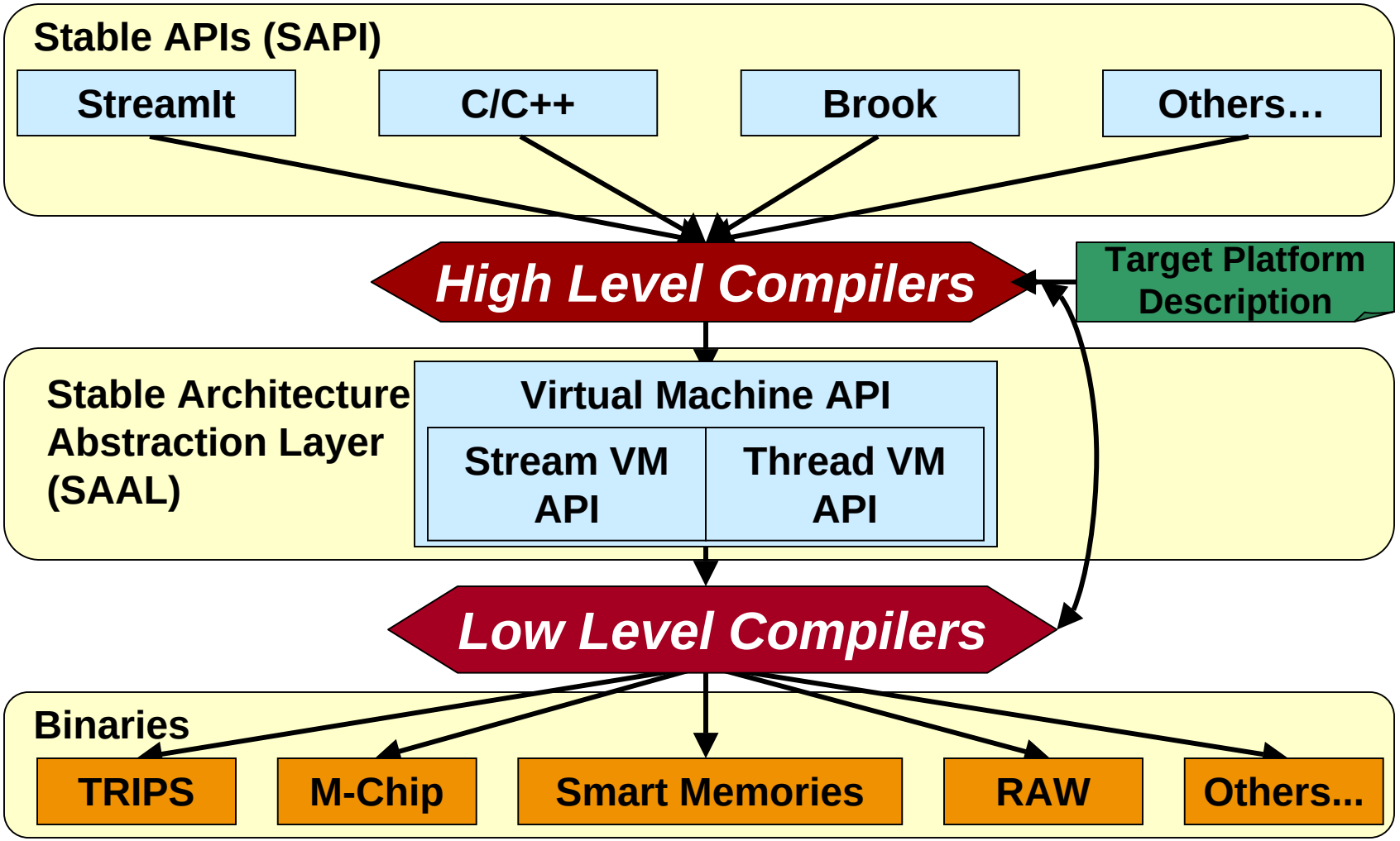
- PCA Hardware is complex and changing
- PCA Missions are complex and changing
- Large amount of configuration, constraints, requirements, etc. information in addition to functional requirement
- Extracting and encapsulating this information
  - Increases portability, scalability
  - Facilitates Reconfiguration, Repurposing, Redeployment
  - Is an important goal of most modern software systems

Metadata System

- **Metadata needed throughout the PCA system**
  - Several contexts
  - Consistent method of representation & query preferred
  - Needed to enable processor and compiler developers to progress
- **Current system stores metadata as XML**
  - Metadata is expressed as relational, hierarchical object oriented structure
  - Instantiated as XML
  - Contexts are defined by a Schema and Documentation
  - Accommodates procedural or static representation queries
  - Accessible to wide range of API's, tools, etc.



Use of Metadata



Platform Description Context

- **Needed by HLC to improve VM output**
  - **Helps allow coarse grain partitioning of applications into appropriate sized pieces**
- **Nearly complete, minor fixes remain**
- **Describes target platform using common dictionary of virtual resources and attributes**
  - **Processors: type, frequency, max-IPC, latency…**
  - **Memories: type, size, cache-linesize, associativity…**
  - **Net-Links: senders, receivers, latency, bandwidth**

Dynamic Configuration

- **Model so far good for flexible resources, goals & constraints**
  - Two level compile
  - structured VM code
  - Well defined metadata
  - Good compilers
- **Dynamic resources, goals, & constraints much harder problem**
  - Builds have (nearly) infinite time to analyze & search the solution space, run-time changes must happen quickly
  - Static, configurable build parameters a hard, but tenable task
  - Support for dynamic criteria explodes the solution space

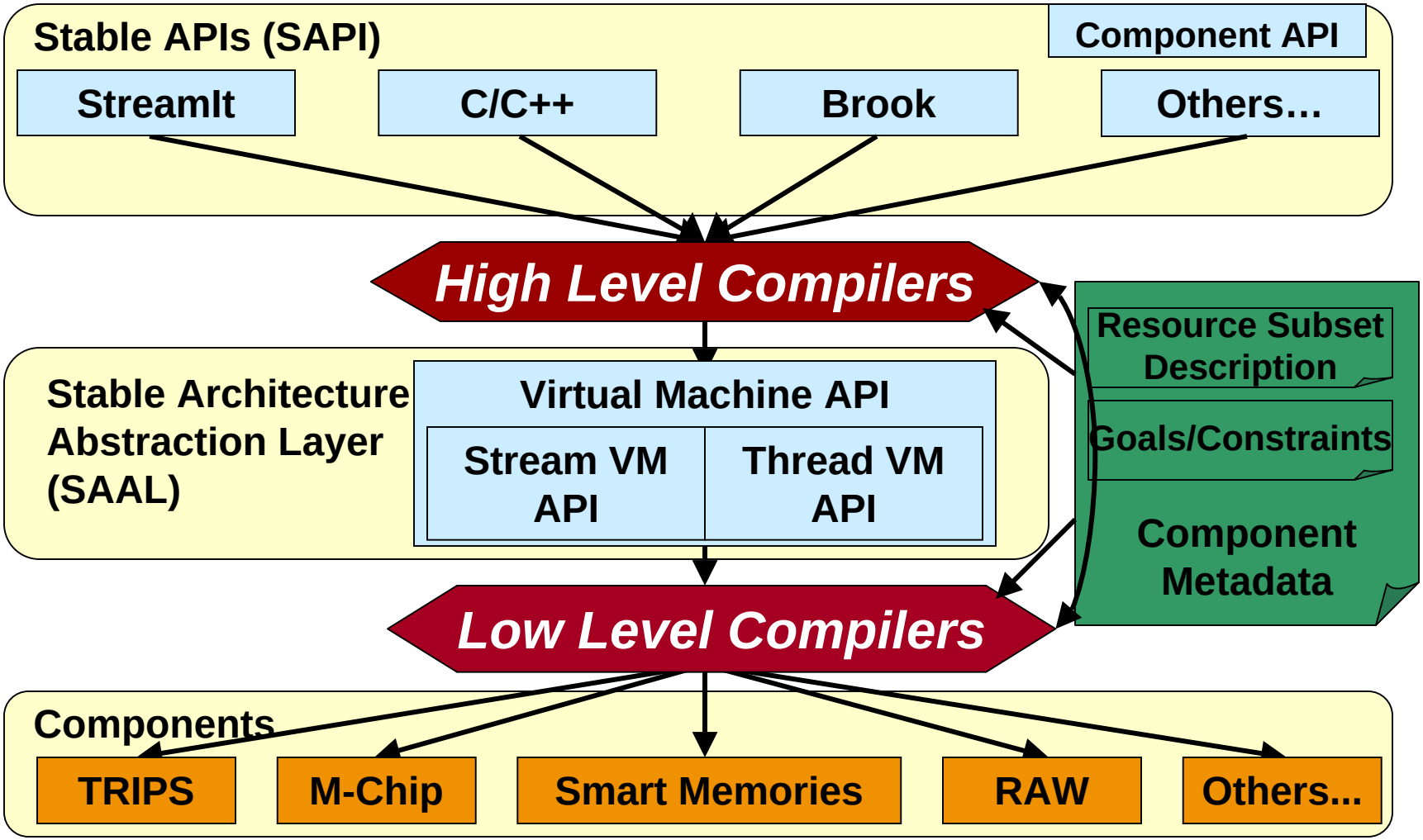
Alternate Monoliths

- **Build with several parameters**
  - Traverse build chain with a defined set of constraints, goals, resources expected
  - Deploy binaries for each set
  - Select the best-fit binary at run-time
- **Benefits**
  - Build chain sooner
  - Easier problem, faster builds
  - Known, testable states
  - Better optimization for known states
- **Problems**
  - Problems with unexpected hardware states
  - Not as flexible as the hardware
  - Only optimal for expected states

Component-Based Approach

- Flexibility & resilience gained by partitioning physical resources
- Configure each partition independently
- Build binaries for each partition in various states
- Benefits:
  - Smaller problem makes flexible build criteria more feasible
  - Hierarchical approach factors the problem of resource management
  - Able to match run-time needs more closely
  - Able to achieve top performance in more situations
  - More easily respond to hardware failures & changes
- Problems
  - Requires a more robust run-time system to fully exploit
  - Many states possible – complicates testing
  - Framework bloat

Component-Building



Morphware Forum Steps

- **Priority: End-to-End framework that allows an application that can reconfigure it's platform**
- **Immediate priorities:**
  - **Finish TVM**
  - **Finish HWMD**
  - **Define HLC / LLC Interaction**
  - **Determine run-time services**
    - **Load, unload, configure, measure, etc.**
  - **Consider component-based approaches**
- **Continue regular activities**
  - **Quarterly meetings, interims, draft documents, etc**

www.morphware.org

- The Morphware Forum web site provides some public information
  - Selected public papers & briefings
  - Links to PCA project sites and related links
  - Link to DARPA PCA
  - This paper and presentation, soon
- In the future, it will provide one-stop public dissemination of MSI documents